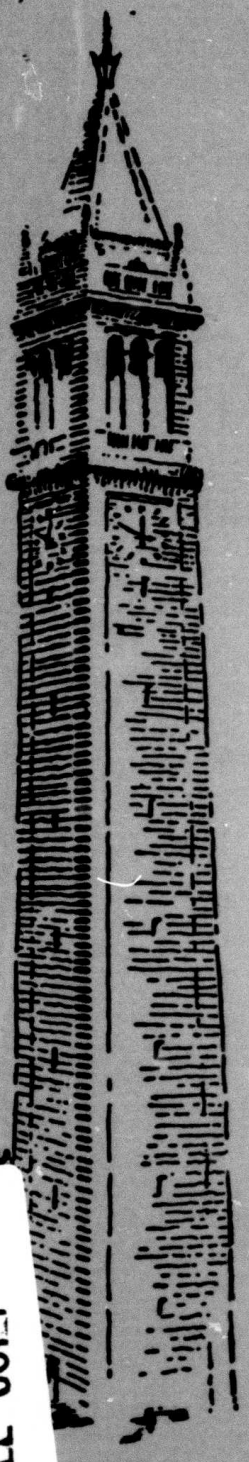

FINAL REPORT

"THE VOCODER PROJECT"
1 JULY 1979 through 31 JANUARY 1983

PRINCIPAL INVESTIGATOR

R.W. BRODERSEN
(415) 642-1779

SPONSORED BY

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY (DOD)
ARPA ORDER NO: 3793

MONITORED BY DEFENSE SUPPLY SERVICE
UNDER CONTRACT NO: MDA903-79-C-0429

January 1983

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley, CA 94720

1. THE VOCODER PROJECT - SYSTEM DESCRIPTION AND SOFTWARE

1.1. Goals of the Project

Since the invention of the first vocoder system a half century ago, a steady trend towards lower-cost and more compact realizations of speech analysis and synthesis systems have been evident (1). The vocoder we designed at UCB was a result of the desire to integrate the entire function of such a system into a single MOS/LSI circuit. At the same time, the opportunity was used to develop a more general design philosophy to facilitate the implementation of other signal-processing functions with the same technology. We created a library of standard circuit functions, the design of which reflects the direction we took in terms of system organization and processor architecture. Essential to this effort was the availability of a modest but well-thought-out collection of computer-aided design tools, and a close association between ourselves and the designers of these tools.

1.2. System Description and Configuration

Vocoder systems encode and decode speech at low bit rates by taking advantage of the short-term stationarity of speech. Generally an interval of time known as a frame is established, over which time the speech signal is assumed to be stationary. Frames tend to be from 10 to 25 ms in length. For each frame a set of parameters is extracted from the input speech by the speech analyzer, and for each set of parameters a frame of synthetic speech may be generated by the speech synthesizer. The set of parameters for the frame typically consists of between 30 and 150 bits of data. The DOD low-bit-rate speech transmission standard specifies 22.5 ms frames consisting of 54 bits each. For a typical 8 kHz sample rate the frame is therefore 180 samples in length.

Our vocoder circuit is intended for use in a system similar to Fig. 1.1. Input and output speech data are transmitted in digitized form to and from the vocoder through a 12-bit parallel bus called the "speech data bus" (S bus). This data is transferred each sample period. Parameter data is through an 8-bit "parameter data bus" (P bus). Finally, a serial stream of parameterized data is transferred from the host microcomputer and a serial port.

Although virtually the entire vocoder algorithm is implemented by the vocoder circuit, the host is responsible for putting the parameter data into the required format for transmission and for performing the inverse operation for received data. This consists mostly of applying coding tables to the data such that the bit rate is reduced. So while the bit rate between the host and the vocoder might be 10,000 bits/sec in each direction, the bit rate between the host and the external serial link is typically 2400 bits/sec.

A second function of the host is to interpolate and filter the data being transferred to the vocoder. In a typical configuration the frame length for transfers between the host and the vocoder might be 1/2 or 1/4 the frame length for transfers between the host and the serial interface. The quality of the vocoded speech is improved if simple interpolation algorithms are used to interface between the two frame rates. The frame rate for transfers between the host and the vocoder may be set by the host, and is referred to herein as the "internal frame rate."

The total amount of processing involved is compatible with a medium-level single chip microcomputer with 2k of internal ROM space for program and table look-up.

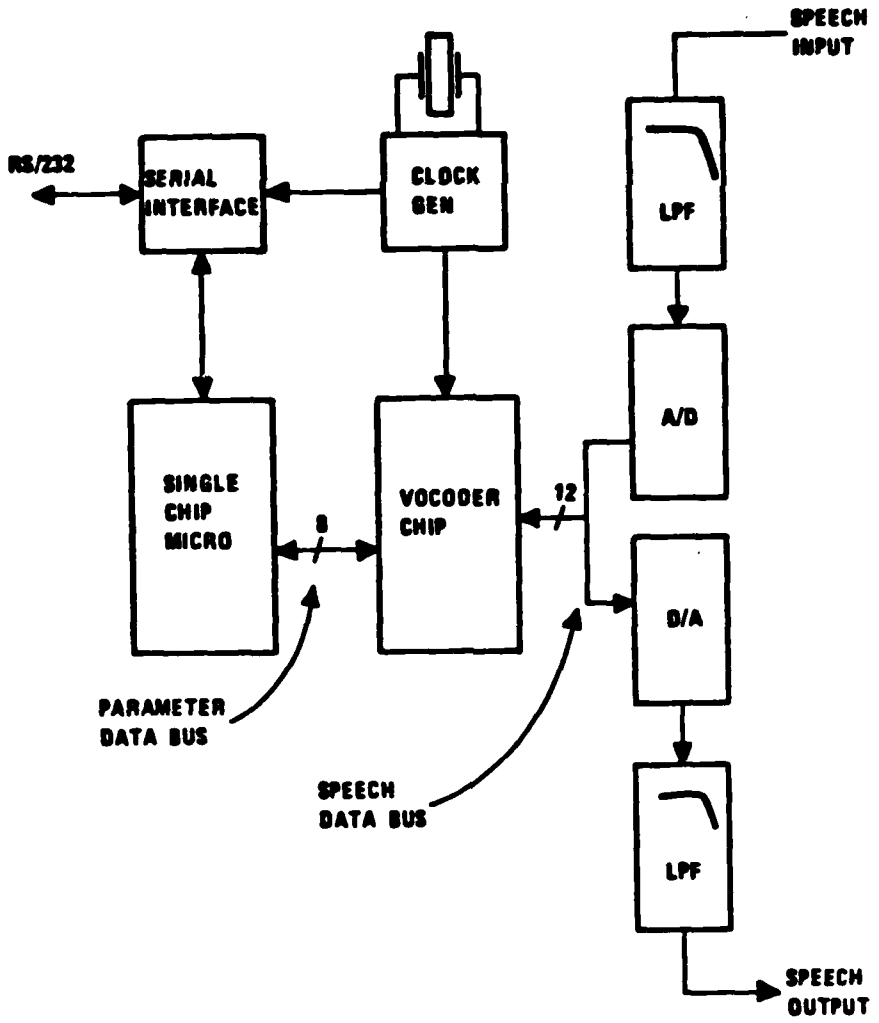

Figure1.1 System configuration

1.3. Algorithms

1.3.1. Spectral Analysis

Vocoder algorithms are generally classified according to the method used for spectral analysis. This portion of the vocoder system generally requires the most computation, and also largely determines the quality of the vocoded speech. Of the several methods possible for implementing a linear prediction analysis of a speech signal, the adaptive lattice algorithm was chosen as being the most appropriate for an LSI implementation. This algorithm, described among others by Kang (2) (who calls it the flow- form LPC algorithm), exhibits the following advantages: 1) As an adaptive algorithm, it does not require the storage of a frame's worth of input data, as would be needed by a block-method analysis; 2) Its adaptation characteristics are regarded as superior to other adaptive algorithm, such as gradient methods; 3) The arithmetic precision

required for the computation is not as great as other methods, particularly those involving matrix inversion, making the use of floating-point arithmetic unnecessary; 4) The lattice analyzer yields reflection coefficients directly, which are the preferred form for low-bit-rate transmission and the specified form for the DOD speech transmission standard; 5) The residual signal is generated automatically by the analysis filter, facilitating use in voice-excited or baseband-encoded vocoder schemes.

In the analysis the speech signal first passes through a preemphasis network Fig. 1.2, improving the distribution of the reflection coefficients, which are otherwise highly skewed (2). This is followed by 10 identical filter stages, detailed in Fig. 1.3. Each stage contributes one zero to the analysis filter.

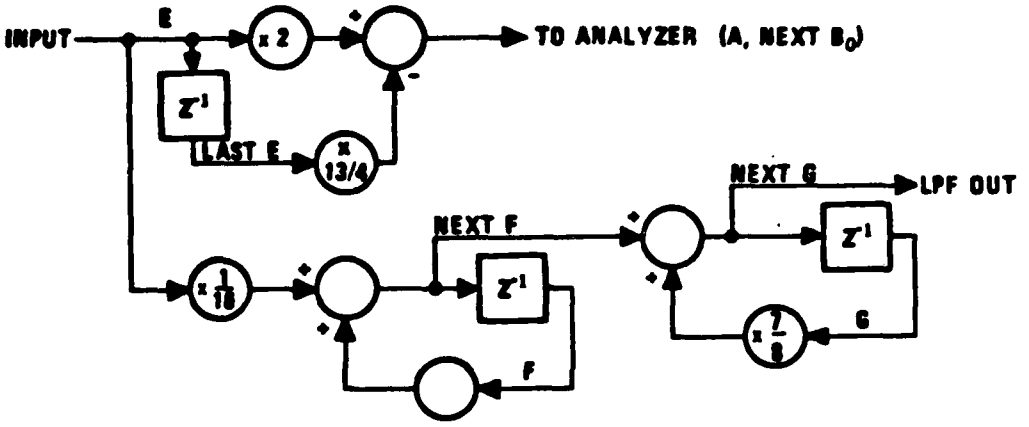

Figure 1.2 Preemphasis flow chart

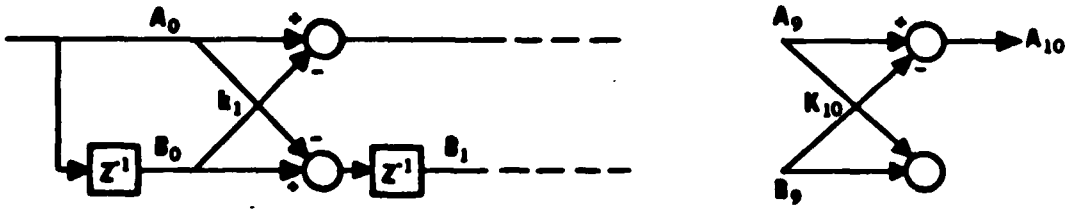

Figure 1.3 The ten-pole lattice analyzer

The precision requirements for the calculations involved can be understood by noting that there are three types of data being handled: the reflection coefficients, which are dimensionless; data that is dimensionally equal to the

speech signal, including the A and B signals; the C and D signals (see Fig. 1.4), which are dimensionally speech squared.

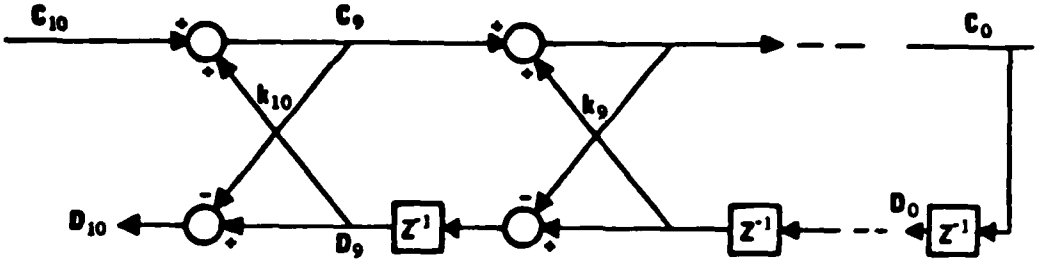

Figure 1.4 The ten-pole lattice synthesizer

High level simulation of this algorithm indicates that representing the reflection coefficients as eight bit quantities does not degrade the spectral accuracy significantly. In order to achieve a dynamic range of 30 dB without degradation, and 45 dB with only minor degradation, simulation indicated a required precision of 16 bits for the A and B signals/, and 26 bits for the C and D signals. The hardware design reflects these varying precision requirements for the different data.

1.3.2. Pitch Analysis

The diversity of pitch detection and voicing decision algorithms/ is even greater than that of spectral analysis. The algorithm used in our system is the Gold Pitch Tracker described by Gold and Rabiner (3) (the particular algorithm used is referred to as the "first modification in (3)). In this approach, the input speech is first low-pass filtered, and then applied to a peak detector. From this, six signals are formed, each a combination of the current sample, previous peak and previous valley of the low-pass filtered waveform. These six signals are fed into six identical elementary pitch detectors, which are a type of phase-locked-loop circuit that attempts to track its input and produce an estimate of the period. Thus six estimates of the pitch are produced. To generate a single pitch estimate, these six estimates are compared by a so-called "scoring algorithm," which selects one of the six as the most likely estimate of the pitch period. If there is not sufficient agreement among the six estimates the speech is considered unvoiced - being noise-like rather than periodic.

The low-pass filter used consists of two cascaded single pole sections. It was found convenient to schedule the execution of the scoring algorithm so that a score is determined for one of the six pitch estimates each period of the input speech. Thus, every six sample periods new pitch/voicing determination is made.

The simple scoring method given for this algorithm in (3) is susceptible/ to occasional single-frame pitch errors, which degrade the quality of the vocoded

speech. Although a more elaborate scoring algorithm is given which improves the situation, the single frame errors may be easily filtered out on a frame-by-frame basis. This filtering is not implemented in the vocoder circuit/ but may be included in the host microcomputer. The filtering is composed of the following substitutions: 1) A single voiced frame in an unvoiced segment is replaced by an unvoiced frame; 2) A single unvoiced frame/ in a voiced segment is replaced by a voiced frame with the pitch period being the average of the previous and following/ frames; 3) The pitch period of a voiced frame is constrained to be between the values for the previous and the following frames, inclusive; 4) The pitch period of a frame following an unvoiced/voiced transitions/ replaced by that of the following frame; 5) The pitch period of a frame preceding a voiced/unvoiced transition is replaced by that of the previous frame.

1.3.3. Synthesis

The synthesizer portion of a vocoder system consists of two functions: An excitation generator, which simulates spectrally flat voiced and unvoiced excitation waveforms; and a synthesis filter, which imparts the desired spectral shape to the signal.

It is common practice to use some waveform other than a simple periodic pulse as the voiced excitation. The waveform must/ be reasonably flat over the frequency range of interest. Since the human ear is nearly completely phase-deaf, an all-pass waveform will sound nearly identical to a simple pulse. However, the all-pass waveform is less likely to overload the synthesis filer and hence contributes to dynamic range.

Both a pulse and an all-pass waveform have a DC component that needs to be subtracted out to allow smooth voiced/unvoiced/ transitions. A high pass waveform avoids this requirement. It is used an unvoiced excitation that has a pulse each sample whose sign is the output of a pseudo-random generator.

The synthesis filter is the all-pole lattice filter shown in Fig. 1.4. A new set of reflection coefficients is used each frame. These coefficients are updated asynchronously/ at the beginning of each frame, resulting in a smooth synthesis if the internal frame rate is less than about 10 ms.

1.4. Microprogram Description

1.4.1. General Description

In this section the various programs which make up the microcode are discussed.

The microprogram consists of the following seven segments:

1. Filter main program
2. Filter subprogram
3. Correlator main program
4. Correlator subprogram
5. Pitchtracker main program
6. Pitchtracker subprogram
7. Miscellaneous signals

These segments are represented, respectively, by six source programs: fmain.s, fsub.s, cmain.s, csub.s, pmain.s, psub.s; and one raw data file: unrom.4.r. Along with ten raw data files that specify the decoder programming for the control roms (decoder.1.r, etc.), these seven files are translated by an

assembler ("assemble.sh") into the following layout files of the programmed ROM's.

rom.1 - filter main program 1/2
rom.2 - filter main program 2/2
rom.3 - filter subprogram 1/2
rom.4 - filter subprogram 2/2
rom.5 - correlator main program 1/2
rom.6 - correlator main program 2.2
rom.7 - correlator subprogram 1/2
rom.8 - correlator subprogram 2/2
lrom.1 - pitchtracker subprogram 1/3
lrom.2 - pitchtracker subprogram 2/3
lrom.3 - pitchtracker main program 1/3
urom.2 - pitchtracker main program 2/3
urom.3 - pitchtracker main program 3/3
urom.4 - miscellaneous signals
decoder.1 - filter, correlator main program 1/2
decoder.2 - filter, correlator main program 2/2
decoder.3 - filter, correlator subprogram 2/2
decoder.4 - filter, correlator subprogram 2/2
ldecoder.1 - pitchtracker subprogram 1/3
ldecoder.2 - pitchtracker subprogram 2/3
ldecoder.3 - pitchtracker subprogram 3/3
udecoder.1 - pitchtracker main program 1/3
udecoder.2 - pitchtracker main program 2/3
udecoder.3 - pitchtracker main program 3/3 miscellaneous signals

The assembler "assemble.sh" performs the following operations:

1) Calls the individual assemblers "algor," "casm," and "pasm" for the filter, correlator, and pitchtracker respectively.
2) Runs editor scripts to break up the assembled output according to how the roms are segmented in hardware.
3) Runs "romgen" scripts to generate the layout of the roms.

The individual assemblers translate each line of the source file into a line in the output file. The source file line is broken into fields, the first of which is always a colon. Subsequent fields are compared to entries in a table and result in the setting or resetting of the appropriate bits in the output word. If a field is recognized as specifying a memory operation, the subsequent field is expected to be the memory address.

1.4.2. Filter Program

The filter main program "fmain.s" implements preemphasis and demphasis/ functions on the input and output speech, and a two pole low-pass filter whose output is sent to the pitchtracker. These functions are described by the flow charts in Fig. 1.2 and 1.5. The filter subprogram"fsub.s," executed ten times per speech ample, implements the ten-pole lattice analyzer in Fig. 1.3 and the ten-pole lattice synthesizer in Fig. 1.4. Signal names in these flow charts correspond to symbolic addresses in the source programs.

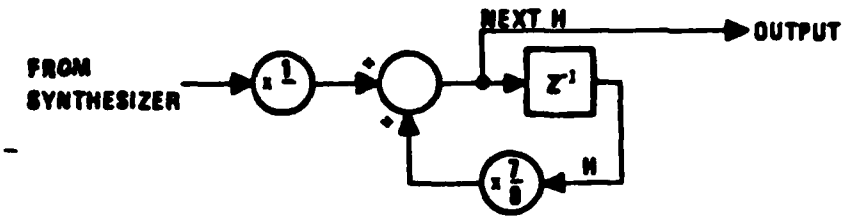

Figure 1.5 Deemphasis flow chart

Symbolic Address	Memory Location	Symbolic Address	Memory Location
A	11	D2	21
B0	9	D3	22
B1	8	D4	23
B2	7	D5	24
B3	6	D6	25
B4	5	D7	26
B5	4	D8	27
B6	3	D9	28
B7	2	D10	29
B8	1	D11	30
B9	0	E	17
B10	31	H	15
C	18	F	14
D0	19	G	13
D1	20		

In addition to the above symbolic address names, the prefixes "next" and "last" are used to refer to locations as addresses during the next or previous speech sample (see description of the Filter Address Generator). For example, "next-B(0)" and "last-A" both refer to location 10. In the source listings below "R N" means "read from memory location N." "W" means "write." "RX N" means "read from memory location (N + contents of index register)" (the index register is incremented each time the subroutine is called). If a signal name (or names) occur(s) at the end of the assembly instruction, the corresponding control line(s) will be active during that instruction cycle (refer to hardware documentation). As an example of the code fmain.s is shown below.

```
fmain.s
: sr:=sr/2 acc:=ks*sr+acc
: sr:=sr/2 acc:=ks*sr+acc
: R A acc:=ks*sr+acc
: W D(1) writelatch acc:=mem
: RC sdben
: RH sr:=mem/2 acc:=mem
: WD(0)sr:=mem/2 acc:=sr writelatch
: acc:=sr+acc sr:=sr/2 in writelatch
: W E acc:=sr+acc sr:=sr/2
: acc:=sr+acc
```

```
: writelatch sdben
: R E
: R last-E acc:=men
: R last-E acc:=acc sr:=mem/2
: R last-E acc:=sr+acc sr:=-mem/2
: sr:=-mem/2 acc:=sr+acc
: R E sr:=sr/2 acc:=sr+acc
: R E sr:=mem/2 acc:=sr+acc
: sr:=mem/2 acc:=sr+acc
: W next-H acc:=sr+acc
: W A writelatch
: W next-B(0)
: R F
: sr:=-mem/2 acc:=mem
: R E sr:=sr/2 acc:=acc
: sr:=mem/2 acc:=sr+acc in writelatch
: W C sr:=sr/2 acc:=acc
: R G acc:=sr+acc
: W next-F writelatch sr:=mem/2
: acc:=sr+mem sr:=sr/2
: acc:=sr+acc
: W next-G writelatch out2
: exit
```

1.4.3. Correlator Program

The correlator program is executed 11 times per sample; only 10 of these are needed, corresponding to one calculation of the normalized cross-correlation at each stage in the lattice analyzer. Figure 1.6 is a signal flow chart of the correlator program. Two memory locations, C and D, are used, for a total of twenty locations for the correlator processor.

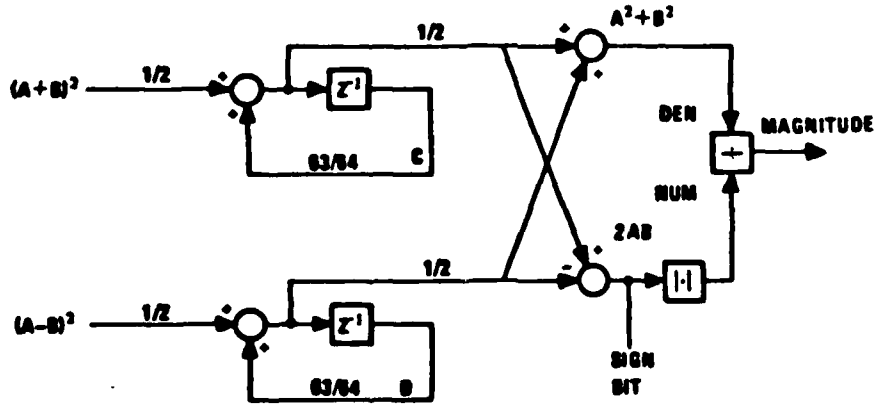

Figure 1.6 Correlator flow chart

1.4.4. Pitchtracker Program

The pitchtracker main program, executed once each sample, does the following:

1) Detect peaks and valleys in the input.
2) Maintain values for "last peak" and "last valley."
3) Form six signals, signal(0)-signal(5):

signal(0) = input
signal(1) = input
signal(2) = input - (last valley)
signal(3) = input + (last valley)
signal(4) = input - (last peak)
signal(5) = input + (last peak)

4) For each sample, compare "score" for current candidate to "top score" and if larger, update "top score" and "winner."
5) Every six sample, compare "winner" to voicing threshold, and if larger, place it in output buffer, if not, clear output buffer. Also clears "topscore."

The pitchtracker subprogram is executed six times per sample, operating mostly on indexed variables. Suppose then that "i" is a value of 0 to 5 representing the iteration of this subprogram. The following operations are performed:

1) Increment the pitch period counter PPC(i), and compare with blanking interval.
2) If greater, decay threshold and do step 3 below; otherwise skip step 3.
3) If signal(i) is a peak for i=0, 2 or 4, or a valley for i = 1, 3 or 5, and signal(i) is greater than threshold(i), then start a new pitch period for the i-th pitch detector by setting:

- threshold(i) = signal(i)
- last pitch period = current pitch period
- current pitch period = pitch period counter
- pitch period counter = 0.

4) Compare the "current candidate" with "pitch period," "last pitch period," and ("Pitch period" + "last pitch period"). In each case, if they agree to within "window" tolerance, increment "score." The "current candidate" is the "pitch period" from one of the six pitch detectors and changes each sample.

In this way the program computes "score" for one of the six pitch detectors each sample, and compares the six scores every six sample and determine a new winner, which is then placed in an output buffer. The output buffer is transmitted each sample.

I.4.5. Refinements

Although the vocoder algorithm used is suitable for most applications there is potential for improvements in several aspects, which are detailed below.

1) Estimation filters: As described above, each reflection coefficient is the normalized cross-correlation of two signals, and this correlation is estimated by a single-pole low-pass filter with a corner frequency of 20 Hz. An improvement in adaption properties is achievable if a two-pole estimation filter is used, with both poles at 53 Hz.

2) Reflection coefficient selection: The adaptive lattice algorithm calculates a new set of reflection coefficients every sample period. The current algorithm simply decimates this data to the internal frame rate prior to encoding and

transmission. Better performance may result if the coefficients are averaged with a rectangular window or selected when the residual energy is at a minimum. Listening tests indicate that the improvement is more pronounced for the case of the two-pole, higher-cutoff estimator described above.

3) Pitch tracker: The pitch tracker algorithm used works well for male speakers but cannot track the higher fundamental frequency in female voices. Techniques presented in (3) could be incorporated to eliminate this deficiency.

1.5. References

(1) Flanagan et al.: "Speech Coding," IEEE Trans. on Communications, Vol. COM-27, No. 4, April 1979, pp. 710-737.
(2) G. S. Kang: "Application of Linear Prediction Encoding to a q Voice Digitizer," NRL Report 7774, October 31, 1974.
(3) Gold, Rabiner: "Parallel Processing Techniques for Estimating Pitch Periods of Speech in the Time Domain."

2. THE VOCODER PROJECT - HARDWARE

2.1. Technology

The vocoder system is implemented as an N-channel, depletion-load MOS circuit of dimensions 225x265 mils (5.7x6.7 mm). The chip contains 27,000 transistors. Topological design rules are those of Mead & Conway (1), modified to be more conservative for metal contacts and to allow buried contacts. Rules regarding the placement of storage nodes were added to guard against dynamic hazards. DC electrical rules are essentially those of Mead & Conway.

AC electrical rules were also developed. The delay time through any critical path must be less than 2/3 of what is needed for functionality, when estimated by Spice simulation using a set of "quasi-worst-case" parameters. This set of circuit parameters is characterized by worst-case values for transistor conductance factor, conductor sheet resistance and field capacitance, and by typical values for threshold and supply voltages, body doping, junction depth and lateral diffusion.

2.2. Design Approach

The most important organizational decision for a design such as this, is how the rather diverse collection or functions that comprise the algorithm should be mapped onto hardware elements. At one extreme, the hardware could be partitioned to look like a signal flow graph of the algorithm, with each stage of the processing implemented with an independently operating circuit. At the other extreme, the hardware could consists of a single processor with sufficient functionality and speed that it may be programmed to implement the entire algorithm. The first approach would suffer from inflexibility and a lack of regularity, making the design task more difficult. The second approach would make it hard to take advantage of the potential for concurrent execution of the different functions. The single processor would also have to be powerful: LPC vocoders have been successfully implemented on three semi-custom MOS/LSI processors (2).

The approach taken here is midway between the above two extremes. The main features are:

1) The bulk of the processing is done by three processing units, identical in design but differing in details of configuration such as word length and size of

data memory.
2) Remaining processing is done by a small collection of custom elements.
3) The three main processors, as well as the more specialized circuitry, are controlled by a single control sequencer.

This approach has several advantages. The main processor design is kept simple, and therefore is both compact and has high performance, since it is not required to be capable of implementing all aspects of the algorithm. The remaining special function circuits account a relatively small percentage of the total area of the circuit. They were therefore designed using a small library of cells (designed by us), rather than in a full-custom fashion, without a large area penalty. The shared control circuitry provides an area advantage that would not be available if the entire Vocoder was not fully integrated.

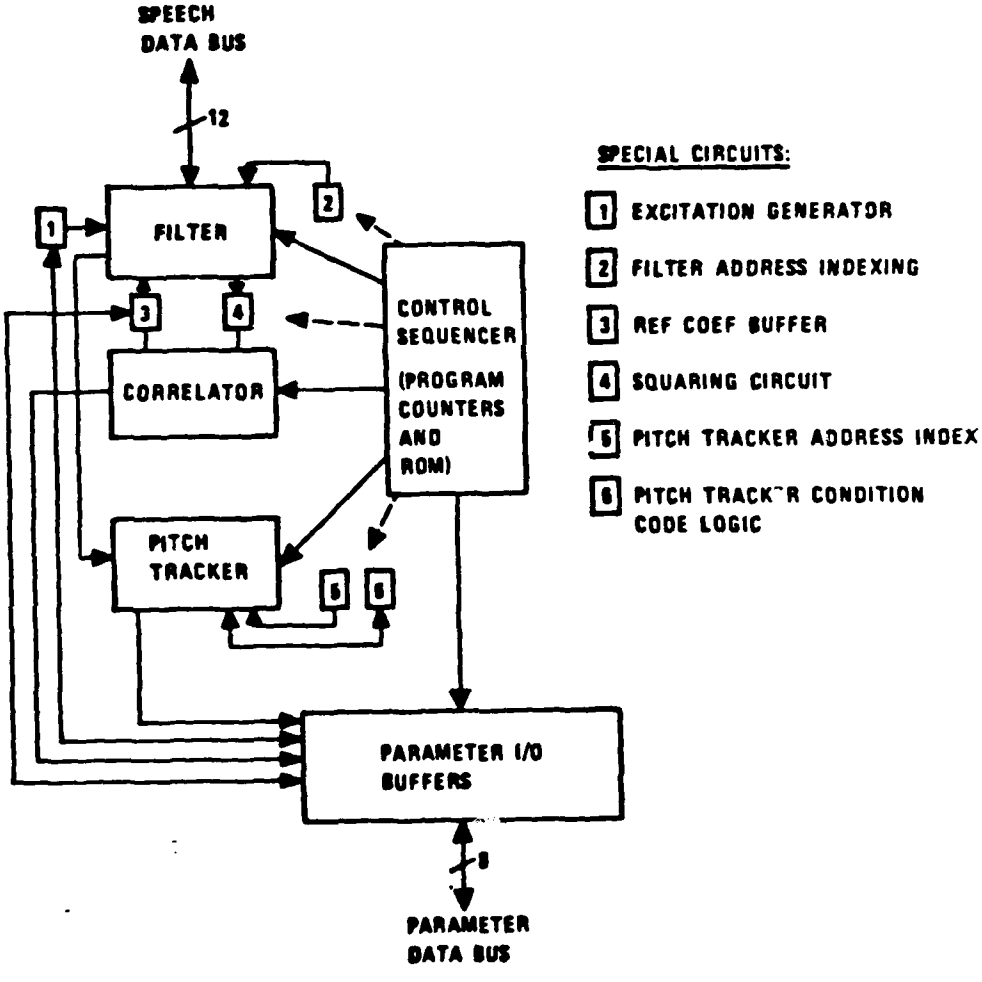

Figure 1.7 Hardware block diagram

2.3. Function of the Different Blocks

Figure 1.7 presents the blocks of the circuit, minus the bonding pads and their drivers. The three main processors are termed the Filter, Correlator and Pitch Tracker. The function of these and of the other blocks of the circuit, in terms of the algorithm previously described, will be discussed in what follows. All data between blocks is transferred bit-serially, except as noted. *Filter:* This processor performs linear filtering on the speech signal. This includes preemphasis and the inear portion of the analysis filter; the synthesis filter and demphasis/; and the low-pass filter that is the first part of the pitch tracker algorithm. The word size in this processor is 16 bits, with a data memory of 32 words. Data from the external Speech Data Bus (SDB) is transmitted in parallel to this block.

Squaring Circuit: The function of multiplying speech data by speech data is replaced by a squaring/ operation. This circuit uses shifters and a look-up table to approximate the squaring operation. The output o this circuit is fed into the Correlator through a parallel port.

Correlator: This processor implements the estimation low-pass filters and some division operations. The word size is 26 bits, and the data memory contains 20 words.

Excitation Generator: This circuit generates the excitation signal needed for synthesis as described in Section 1.3.3. The output signal is fed into the Filter through a parallel port.

Pitch Tracker: This processor implements the Gold pitch-tracking algorithm, including peak detector, elementary pitch detectors and scoring algorithm. The word size is 18 bits, with a data memory of 42 words. Five of these memory locations are read-only constants.

Filter Address Generator: Implements a base register, index register, and address arithmetic for the Filter processor.

Pitch Tracker Address Generator: Performs a similar function for the Pitch Tracker.

Reflection Coefficient Buffer: Stores the reflection coefficients calculated by the Correlator until they are needed by the Filter processor for the analysis filter.

Condition Code Circuit: Unlike the Filter and Correlator, the Pitch Tracker needs conditional operations to implement such functions as peak detection and determining the beginning of a pitch period. This circuit consists of three condition code flags and a small PLA-based finite state machine. Instead of the more typical conditional branch instruction, a conditional write operation is employed, which performs a write operation to data memory only if the condition code flag is set.

Parameter 10 Buffer: Consisting mostly of register arrays, analysis and synthesis parameters are stored here on their way to and from the host microcomputer. Also part of this circuit is a frame counter programmed by the host, determining the number of samples in each internal frame.

Controller: Program counters and read-only memory, providing control and timing signals for all other circuits. The processors and other circuits require in

general horizontal control words, totaling 80 bits for all circuits. The total amount of read-only storage is about 5200 bits.

2.4. Processor Architecture

Initial investigations into the possibilities for signal processor architecture led to the following conclusions:

1) Pipelining should be used to allow concurrent arithmetic operations and data memory operations.

2) Pipeline segmentation within the arithmetic unit should be the minimum necessary to achieve full arithmetic unit utilization to avoid latency in the data path.

3) Parallel/serial multiplication is efficient for most applications; full parallel multipliers consume large amounts of area and power, while bit-serial multipliers require excessive amounts of control circuits. A single accumulator architecture allows parallel/serial multiplies and divides with a minimum of hardware.

The processor consists of a random-access data memory, an arithmetic unit, and a 10 port section.

The data memory can perform a read or write operation each clock cycle. Dynamic three-transistor cells are used. There is no provision for address arithmetic of any sort; external circuits must generate the address if anything other than direct addressing (such as indexing) is required.

The arithmetic unit consists of four registers, a complementer, an adder, and a certain amount of multiplexing and gating. The memory output register (MOR) is a master-slave register which is loaded each cycle with the output of the data memory. Its output feeds the complementer, which under program control will give the true, complement, or absolute value of its input.

The shift register (SR) is a master-slave register which under program control will either load or shift right (arithmetic). In the first case this register gets the output of the complementer divided by two; in the second case it gets its previous contents divided by two. Thus the possible values that maybe loaded into this register were summarized as follows:

SR:= (MOR)/2
SR:= -(MOR)/2
SR:= |(MOR/2|
SR:= (SR)/2

When performing a parallel-serial multiply or divide operation, this register is used to shift right repeatedly on successive cycles. This presents a sequence of partial products to the adder inputs.

The adder is of the saturating variety: if an overflow is detected, the output will be the maximum positive or negative number that can be represented, for positive or negative overflow respectively. A fractional 2's complement number system is assumed.

The output of the adder is loaded into the accumulator. Given the possible values for the adder inputs, the following summarizes the values that may be loaded into the accumulator:

ACC:= 0

ACC:= (MOR)
ACC:= (ACC)
ACC:= (SR)
ACC:= (SR) + (MOR)
ACC:= (SR) + (ACC)

One additional feature of the accumulator, included to allow for a parallel-serial decision operation, is an "accumulate-if-positive" control; when this feature is used, the output of the adder is loaded into the accumulator only if it is positive.

The output of the accumulator drives the M-bus, which feeds the memory input latch (MIL)> This latch is transparent, and may either be loaded or hold data under program control. All memory write operations store the data in this latch; by holding the latch transparent, the accumulator is written directly into memory.

All 10 transfers are also passed via the M-bus. Provided are parallel input and output ports, two serial output ports, and one serial input port. The fashion in which the arithmetic unit is used to multiply a word in data memory by a signed coefficient is straightforward. The coefficient is fed serially, MSB first, into the appropriate control inputs for the processor. Divide operations are slightly more complex. Two words of positive data stored in the data memory can be divided to produce a quotient between 0 and 1.

2.5. Timing and Control

Discussion of timing and control issues for the vocoder circuit can be divided into two broad categories/: Timing related to the internal frame rate at which parameters are transferred to and from the host microcomputer, and timing related to the speech sample rate. The first category applies to the Parameter 10 Buffer, whose data is updated each frame. The second category applies to the entire rest of the circuit; the choice of adaptive algorithms as opposed to block methods implies that essentially all processing is repreated at the speech sample rate, resulting in simpler implementation for the control sequencer.

To reduce the overhead requirements on the host, it is desirable to transfer all analysis and synthesis parameters as a single block of data once per frame, leaving the remainder of the frame for the host to perform its tasks uninterrupted. This is done by having the Vocoder provide to the host a "End-of-Frame" (EOF) signal which either interrupts, or is polled by the host. The interrupt is acknowledged when the host initiates a data transfer, which occurs asynchronously with internal clocks in the Vocoder. Synthesis parameters are double buffered, i.e., parameters transferred to the Vocoder one frame are used by the synthesizer the following frame. Analysis parameters are loaded into the parameter buffer during the last sample of each frame, to be transferred to the host following the beginning of the subsequent frame. This buffering arrangement is very non-restrictive on both the host and the remainder of the Vocoder. The price paid for this flexibility is that the Parameter 10 Buffer is relatively large and consumes 15% of the chip area.

The remainder of the Vocoder in general performs the same sequence of operations each sample period. Most processing is linked to the operation of the Filter processor so the timing for Filter will be discussed first.

The Filter executes a 32 instruction main program and ten times a 32 instruction subroutine each sample interval. Each time the subroutine is called, the memory addresses may be automatically incremented using an optional indexing addressing scheme.

Each instruction requires a complete 2-phase clock cycle for execution, hence there are 352 clock cycles per 125 microsecond sample interval, giving a system clock frequency of 2.816 MHz. The main program implements preemphasis, demphasis/, and the pitch-tracker low-pass filter. In addition, IO transfer between Filter and the external speech data bus is handled during the main program.

The subroutine implements the synthesis filter and the linear portion of the analysis filter, as described in Section 1.4.2. One pole (or zero) of each filter is implemented by each execution of the subroutine.

Timing considerations for Correlator, Excitation Generator, Squaring Circuit, Reflection Coefficient Buffer, and Filter Address Generator are closely related to the Filter timing. Also, the transfer of reflection coefficients to and from the Parameter IO Buffer occurs in synchronism with the execution of the subroutine by the Filter processor.

In the case of the Pitch Tracker, the sequence of operations is also repeated each sample, but the timing sequence is different. There is a 37 instruction main program, and a 43 instruction subroutine, with the subroutine being repeated 6 times. The main program implements the peak-detector part of the Gold pitch tracker algorithm, while the 6 iterations of the subroutine implement the 6 elementary pitch detectors. Portions of the scoring algorithm are implemented by both the main program and the subroutine. Put together, 295 instructions are executed, meaning that the Pitch Tracker processor is idle for 57 of the 352 cycles.

Sequencing control is dominated by a 9 bit, mod 352 program counter, consisting of a mod 32 counter for the five least significant bits (LSB), and a mod 11 counter for the four most significant bits. This counter addresses the control ROM for all functions except the Pitch Tracker subroutine. The four MSB's of the program counter indicate which iteration of the Filter main program is being executed, and hence are made globally available for indexing purposes. A section of control ROM containing code for the Pitch Tracker subroutine is addressed by a separate subroutine program counter (SUBPC).

CONCLUSIONS

At this time the chip has been completely designed and the individual blocks are in various stages of being evaluated. The complete circuit as shown in Fig L8 has several layout errors as well as program errors. We are planning to make the appropriate corrections and have a complete working chip by the end of 1983.

We feel the design approach used here has great merit and we are now using it in the design of several other chips for speech synthesis and recognition. We plan to continue this work and develop the software tools so that an algorithm developer will be able to design a chip with only a minimal amount of actual layout.

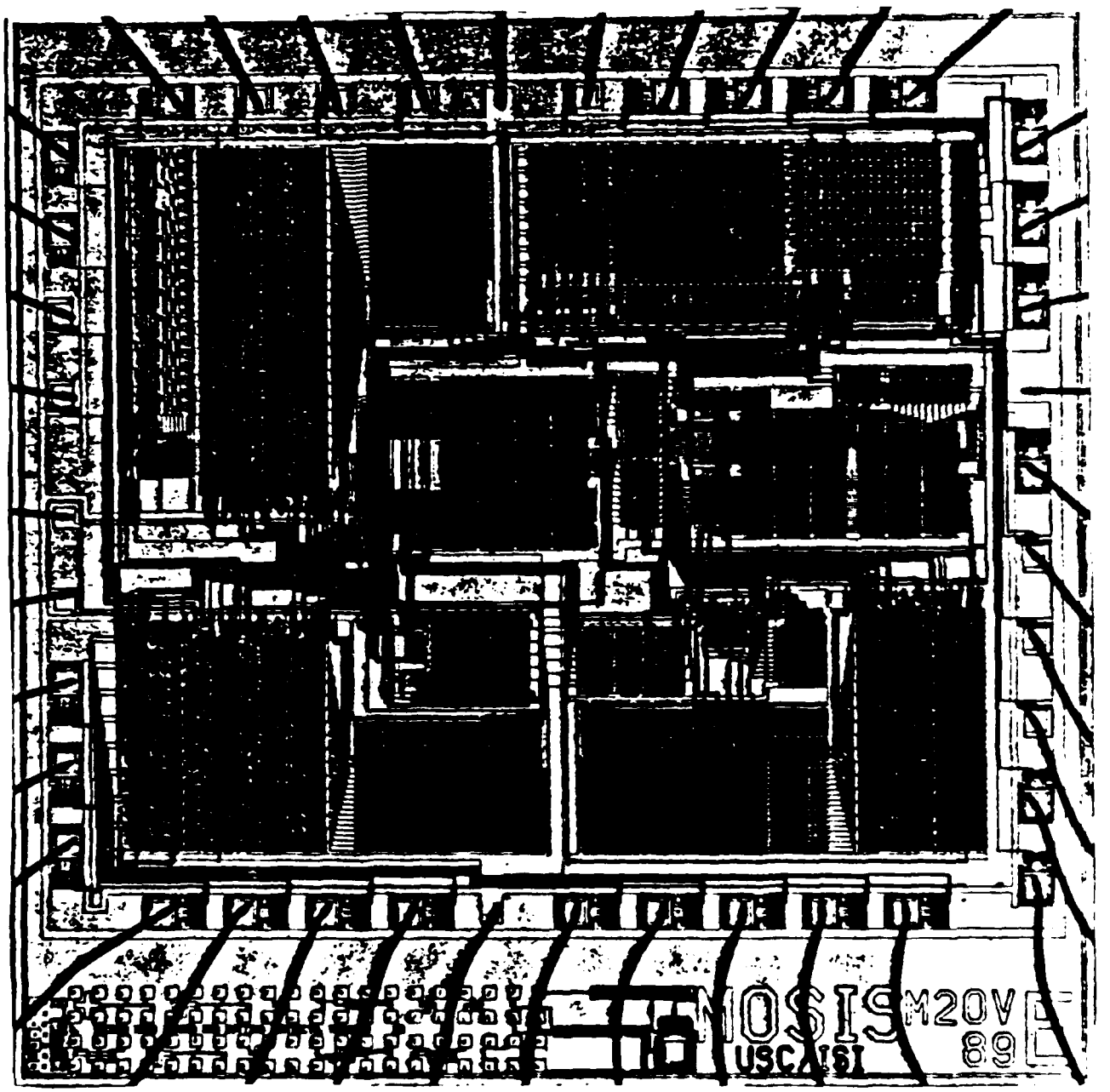

Figure 1.8 The complete chip